1. Notation des TP

Chaque semaine vous allez devoir rendre une partie de votre travail sur Moodle. Par exemple, vous devrez rendre un fichier xml, un source, ou une application entière. Ces documents seront évalués dans le cadre de la note de TP. Ainsi, il n'y aura pas un TP noté final, mais de multiples travaux à faire dans le cadre d'un contrôle continu.

2. Découverte du SDK Android

On va commencer par les outils de développement : le *Software Development Toolkit* d'Android et Eclipse.

2.1. Software Development toolKit : `SDK`

Le SDK Android a été téléchargé sur `C:\appli\info\Programmation\eclipse kepler\AndroidSDK... \sdk` Allez voir ce qu'il y a dedans : plusieurs dossiers, dont :

- `tools` qui contient les outils de gestion du SDK et des AVD. Cependant à l'IUT, vous ne pourrez pas les lancer d'ici directement. Vous devrez passer par Eclipse.
- `platform-tools` qui contient quelques programmes utiles comme `adb.exe` et `sqlite3.exe`.

2.2. Eclipse

Lancez `Eclipse Kepler`. Si le plugin Android n'a pas encore été configuré, vous devrez fournir le chemin du SDK dans les préférences d'Eclipse, onglet Android. Il faut donner le chemin complet qui permet d'arriver au dossier du SDK, là où il y a le sous-dossier `tools`.

Avant de commencer à travailler, on va modifier une préférence concernant l'encodage des fichiers. Ouvrez le menu `Window/Preferences`. Dans cette boîte de dialogue, dépliez `General/Workspace` et tout en bas, dans `Text File Encoding`, choisissez `UTF-8`. Cela va permettre de pouvoir échanger des fichiers avec Unix sans souci d'accents et autres. Vous pouvez aussi choisir `Unix` pour `New text file line delimiter`.

Remarquez les deux icônes dans la barre d'outils, au premier quart gauche. Il y en a une pour lancer le gestionnaire de SDK (`Android SDK Manager`) et l'autre pour le gestionnaire d'AVD (`Android Virtual Device Manager`).

Figure 1: Barre d'outils Android dans Eclipse

2.2.1. Gestionnaire de SDK

Regardez quels paquets ont été installés : les différentes versions d'Android et les outils. Le gestionnaire va automatiquement vous proposer d'installer les mises à jour et la dernière version

du SDK, mais ne le faites pas[1]. Vous ne devrez le faire que chez vous, et à condition de mettre à jour le plugin Eclipse simultanément.

En principe, le SDK versions 15 (Android 4.0.3) et 17 (Android 4.2.2) sont installés.

2.2.2. Android Virtual Device : `AVD`

L'autre gestionnaire sert à créer des émulateurs de tablettes. Vous avez un onglet `Device Definitions` pour les *types de tablettes* et un onglet `Android Virtual Devices` (affiché par défaut) pour les tablettes que vous avez créées.

On va commencer par modifier une définition de type de tablette. Choisissez l'onglet de droite. Cherchez `7" WSVGA (tablet)` dans la liste. Cliquez sur le bouton `Clone...`. Puis dans cette nouvelle fenêtre, changez `Buttons:` de `software` à `hardware`. Changez le nom de ce type de tablette en `My 7"...` puis cliquez sur `Clone`. Voici donc un nouveau type d'appareil. Vous pourriez définir un modèle correspondant à votre smartphone.

Créez une tablette ayant les caractéristiques suivantes :

- Device = votre type de tablette
- Target = Android 4.2.2, level 17
- CPU = ARM, mais sur Windows ATOM serait un bien meilleur choix
- cocher `Hardware keyboard present`
- choisir `Skin with dynamic hardware controls`
- RAM = 1024 Mo (mais c'est votre PC qui décide, y-a-t-il assez de RAM pour Windows, Eclipse et l'AVD ?) et Heap = 32 Mo
- Internal Storage = 128 Mo (pour vos applications et données)
- Carte SD : mettre une taille de 9Mib
- Emulation : Use Host GPU

À l'IUT, vous allez devoir faire une manip bizarre. Les définitions de tablettes virtuelles sont mises dans des fichiers du dossier `.android` de votre compte. Le problème, c'est que vous avez deux dossiers de compte : `Z:\` (fileriut) et `C:\Utilisateurs\vous?????` (copie locale du profil itinérant), les ????? sont soit rien, soit DOMLAN, soit un numéro. De fait, il y a un dossier `.android` dans les deux endroits et ça pose un gros problème : les fichiers sont mis dans l'un, mais cherchés dans l'autre.

- Supprimez le dossier `C:\Utilisateurs\vous?????\.android` et recopiez `Z:\.android` à la place.
- Vous devrez faire cela à chaque fois que vous modifiez la configuration d'un AVD.

Lancez cette tablette virtuelle : bouton `Start...` quand elle est sélectionnée. Dans le dialogue qui arrive, choisissez :

- Si vous êtes sur un petit écran, cochez `Scale display to real size` avec `Screen size` = 16, `Monitor dpi` = 56, ces valeurs permettent de réduire la fenêtre de l'AVD à 75%.
- ne cochez pas `Wipe user data` sauf si vous voulez tout supprimer sur la tablette, y compris vos préférences.

[1] Les mises à jour sont bloquées à l'IUT afin de ne pas désynchroniser le plugin du SDK. Si on laissait faire les mises à jour automatiquement, vous seriez rapidement bloqué(e) dans les TP, plus rien ne se compilerait.

Notez qu'elle met beaucoup de temps à devenir disponible parce que c'est un émulateur de machine, comme VMware. Vous devrez faire attention à ne pas la fermer à chaque fois. Modifiez les préférences de cette tablette. Pour cela, vous avez le bouton `MENU` ou la touche `F2` pour accéder aux préférences. Relire le cours pour le mode d'emploi.

- Mettez-la en langue française
- Enlevez le verrouillage de l'écran en cas d'inaction

NB: faites attention à ne pas mettre cet AVD en veille, car cela ferait créer une image de la RAM de 512Mo dans votre compte et exploser votre quota disque.

2.3. Android Debug Bridge : `ADB`

Avant de passer à la programmation, on va communiquer avec cette tablette virtuelle. Ouvrez une fenêtre `cmd.exe`. Plusieurs commandes sont à essayer. Elles sont expliquées dans le cours.

- `adb devices`
- `adb shell`. Dans cette connexion à la tablette, tapez :
  - `ls` : c'est une arborescence Unix,
  - `cd /data/data`
  - `ls` : voici la liste des paquetages Java des logiciels installés,

  Consulter cette page qui explique comment rendre la carte sd virtuelle inscriptible. Actuellement, elle est en lecture seule.
  - `su`
  - `mount -o rw,remount rootfs /`
  - `chmod 777 /mnt/sdcard`
  - `exit`

  Maintenant, la carte mémoire sd peut être modifiée.
  - `mkdir /mnt/sdcard/Pictures` : crée un dossier sur la carte sd virtuelle, mettez un autre nom s'il existe déjà...
  - `exit` pour sortir du shell et revenir dans Windows.
- `adb pull /init.rc` : ce fichier spécifie la séquence de démarrage de la tablette, comme `/etc/rcS.d` et `/etc/rc2.d` sur Ubuntu
- `adb push C:\Windows\Web\Wallpaper\Landscapes\img7.jpg /mnt/sdcard/Pictures` : envoyez une image quelconque sur la tablette, dans le dossier correspondant au stockage interne sdcard que vous avez créé auparavant. L'image sera visible dans l'application `Gallerie` mais peut-être après un redémarrage de la tablette.

Vous voyez la ressemblance de `adb` avec des outils comme `telnet`, `ssh`, `ftp`.

3. Première application

3.1. Création

Utilisez l'assistant de création d'une application Android pour créer votre première application.

- Choisissez un niveau d'API qui corresponde à ce qui est installé dans le SDK (17),
- Laissez l'icône proposé par défaut, ou alors trouvez une image qui ait la taille demandée,
- Choisissez une application minimale : `Blank Activity`

- Cochez la création de la classe d'activité `MainActivity`.

Eclipse crée plusieurs dossiers et plusieurs fichiers. Allez voir ce qu'il y a dans `src` et `res`. Par exemple, dans `res`, il y a plusieurs dossiers `drawable` qui contiennent différentes versions de l'icône d'application, de diverses tailles pour s'adapter à des écrans différents. Dans `src`, les sources sont groupés par paquetage. `gen` et `bin` sont reconstruits automatiquement dès que vous changez quelque chose.

Sur certains postes, Eclipse crée une application plus complexe. Vous trouverez deux fichiers dans `res/layout` : `application_main.xml` et `fragment_main.xml`, et d'autre part, la classe `MainActivity` est plus complexe.

3.2. Exécution

Le lancement de l'application est simple. La première fois, il faut demander à exécuter l'application en tant que `Android Application`. Les fois suivantes, ça sera mémorisé dans ce qu'Eclipse appelle une *configuration de lancement*, voir le menu déroulant `Launch configurations` qui apparaît quand on clique sur la flèche vers le bas dans la figure 2 ci-dessous.

Figure 2: Bouton Run As...

Ensuite, s'il n'y a aucune tablette connectée ni AVD en cours, Eclipse vous demande quel AVD il faut lancer. Il en faut un qui corresponde au niveau d'API de votre application.

4. Affichage d'un message

Ouvrez le source `MainActivity.java` et trouvez la méthode `onCreate`. Cela doit ressembler au source suivant. Complétez-le avec les instructions qui permettent de faire afficher un message dans la fenêtre `LogCat`.

```
import android.util.Log;

public class MainActivity extends Activity {

    public static final String TAG = "hello";

    @Override
    protected void onCreate(...) {
        super.onCreate(...);

        Log.i(TAG, "Salut\ !");
```

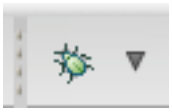
4.1. Mise au point

Nous allons expérimenter quelques aspects du débogueur intégré dans Eclipse. Ce n'est pas spécifique à Android, mais est disponible pour tout projet Eclipse. Ici, on va être un peu limités par la petitesse du programme. On va seulement pouvoir découvrir quelques techniques très utiles.

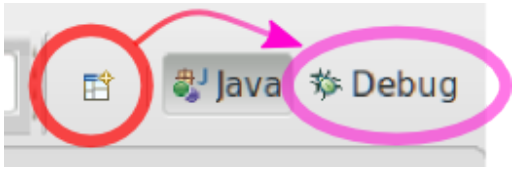
A chaque fois que vous modifiez un source Java, le débogueur vous signale qu'il doit se déconnecter de la tablette. En effet, si vous changez le source, ce n'est plus ce qui est exécuté sur la tablette, donc la mise au point ne peut pas continuer.

4.1.1. Lancement en mode debug

C'est ce bouton qui lance l'exécution en mode mise au point.



En principe, il fait basculer l'interface dans la *perspective* `Debug`. C'est une disposition des fenêtres un peu différente pour faciliter la mise au point. Si elle n'apparaît pas, vous pouvez demander de l'afficher, c'est en haut et à droite, cliquez suir le bouton puis choisissez `Debug` dans la liste des *perspectives*.



Voici les boutons utiles pour mettre le programme au point. De gauche à droite, demander de continuer l'exécution à pleine vitesse jusqu'au prochain point d'arrêt, mettre en pause l'exécution, arrêter totalement le programme, déconnecter `adb`, exécuter une instruction en entrant dans les méthodes, exécuter la ligne suivante sans rentrer dans les méthodes, exécuter le reste de la méthode à vitesse normale puis s'arrêter après son return. Nous n'emploierons pas les autres boutons.

La mise au point repose sur ces deux techniques :

- Lancer en mode normal et attendre que ça se plante. L'exécution s'arrête alors exactement sur l'instruction qui est mauvaise, par exemple celle qui déclenche une `NullPointerException`.
- Mettre un *point d'arrêt* avant l'endroit qu'on pense mauvais puis exécuter ligne par ligne pour voir ce qu'il y a dans les variables et guetter l'erreur.

4.1.2. Placement d'un point d'arrêt

Double-cliquez dans la marge gauche au début de la méthode `onCreate()`.

La fenêtre `Breakpoints` liste tous les points d'arrêts que vous avez placé. N'en mettez pas trop dans un programme, vous aurez du mal à suivre tout ce qui peut se passer. Mettez vos méthodes au point les unes après les autres, pas tout en même temps.

4.1.3. Examen de la pile d'appels

Lorsque le programme s'arrête, soit sur une erreur non prévue, soit sur un point d'arrêt, Eclipse affiche la pile d'exécution (imbrication des appels de méthodes) dans la fenêtre `Debug`, figure 3.

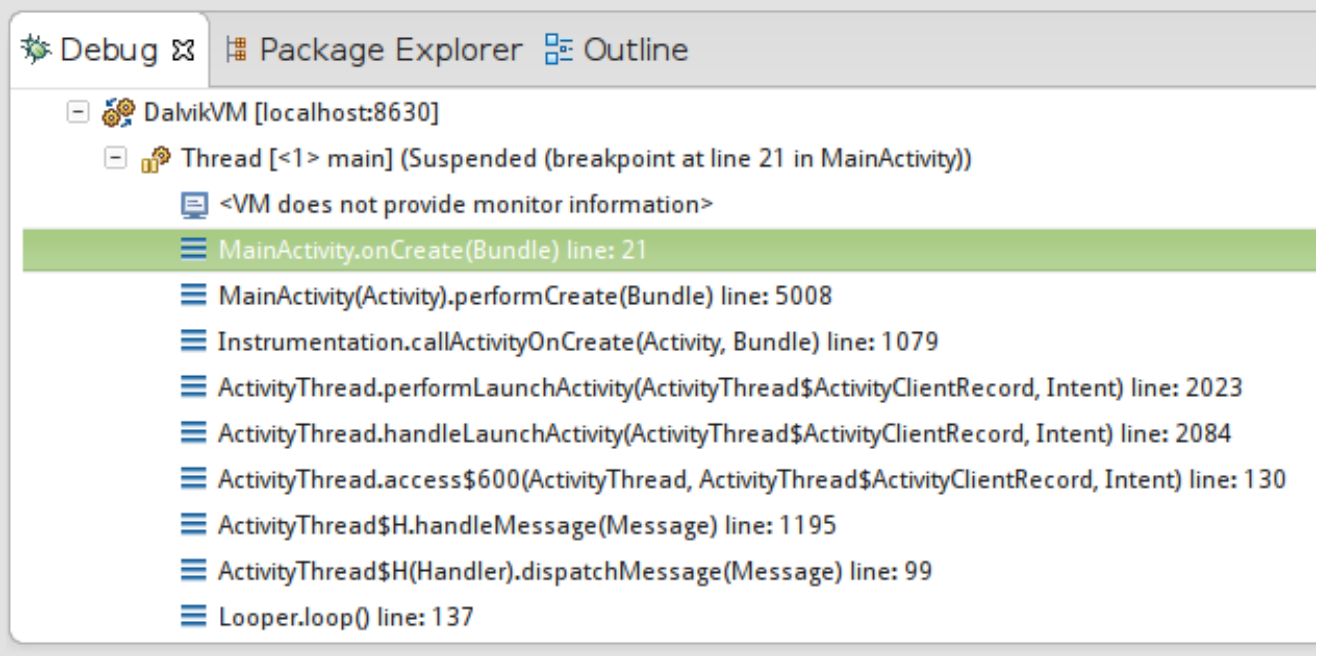


Figure 3: Fenêtre Debug

Une grande partie des appels ne viennent pas de votre application mais du système Android.

Un autre problème est qu'une application Android implique plusieurs *threads*. Parfois, il faut arriver à comprendre quel thread plante. Ce n'est pas forcément `<1> main` comme ici.

4.1.4. Exécution pas à pas

Rajoutez ceci à la fin de la méthode `onCreate()` de `MainActivity` :

```
    int v = 54;
    int n = v / 9 + 3;
    int f = factorielle(n);
    Log.i(TAG, n+"! = "+f);
```

avec ceci un peu plus loin :

```
private int factorielle(int n)
{
    int r;
    if (n > 1) {
        int fnm1 = factorielle(n-1);
        r = n * fnm1;
    } else {
```

```
        r = 1;
    }
    return r;
}
```

Réglez `LogCat` pour n'afficher que des messages `info` au moins.

Mettez un point d'arrêt sur la première instruction, `int v = 54;` puis lancez en mode mise au point.

Puis essayez les modes suivants :

- Ligne par ligne : appuyer sur `F6` (`step over`) pour exécuter la prochaine ligne. Suivez les modifications des variables dans la fenêtre `Variables`. Voyez que l'appel à `factorielle` a été fait à pleine vitesse.
- Instruction par instruction avec la touche `F5` (`step into`) : elle rentre dans les appels aux méthodes. Alors pour factorielle, ça va un moment, mais c'est pénible. Vous pouvez couper court avec `F7` (`step return`).

Supprimez l'exécution en cours à l'aide du bouton `Terminate` (carré rouge dans la barre d'outils).

4.2. Mise en place d'un bug

Normalement, c'est l'inverse qu'on fait. Les bugs, on les dégomme. Là, on va créer un bug et essayer de voir si on peut le retrouver.

Rajoutez ceci juste après `<TextView` dans `res/layout/activity_main.xml` ou `res/layout/fragment_main` ça sera expliqué dans le cours de la semaine 2 : 🡻

```
<TextView
    android:id="@+id/texte"
    ...  />
```

Nettoyez le projet : Menu `Project/Clean...` pour mettre à jour les identifiants.

Puis rajoutez ceci à la fin de la méthode `onCreate` de `MainActivity`, après le `setContentView` : 🡻

```
TextView texte = (TextView) findViewById(R.drawable.ic_launcher);
texte.setText("Ça marche");
```

`R.drawable.ic_launcher` est un identifiant, mais pas celui du `TextView`, alors que va-t-il se passer ? C'est une erreur assez fréquente. Il est important de savoir la réparer.

Vous devez :

1. Configurer `LogCat` pour ne voir que les messages d'erreur, vider `LogCat` avant chaque tentative.
2. Lancer le programme en mode normal, à pleine vitesse et voir le popup « L'application HelloWorld s'est arrêtée ». Qu'est-ce qui est écrit dans `LogCat`, en rouge ? Ça vous dit où ça s'est planté, mais c'est dans une fonction du système `ActivityThread.performLaunchActivity`.

Il faut chercher l'autre explication plus bas : `Caused by: java.lang.NullPointerException` et juste en dessous, la ligne problématique *nom de fichier:n° de ligne* dans votre programme. Vous pouvez double-cliquer sur cette ligne de `LogCat` pour aller dans le source.

3. En fait, en mode mise au point, ça ne s'arrête pas sur la ligne problématique, donc il faut poser un point d'arrêt sur le début de la méthode `onCreate` puis tracer ligne par ligne jusqu'à celle qui pose un problème. Et ensuite, insister avec `continue` pour que les messages d'erreur s'affichent.

Voici maintenant un autre bug, aussi fréquent :

```
Button bouton = (Button) findViewById(R.id.texte);
bouton.setText("Ça marche");
```

Cette fois, R.id.texte est un identifiant, mais ce n'est pas celui d'un `Button`, il va y avoir une erreur de conversion de classe : `ClassCastException`[2]. Là également, le mode debug n'arrête pas l'exécution exactement là où elle se produit.

5. Travail à rendre

Vous devrez déposer `src/*/MainActivity.java` sur le dépôt Moodle à la fin du TP.

[2] `TextView` est une superclasse de `Button` mais on n'a pas le droit de convertir de la superclasse vers une sous-classe. Seul l'inverse est possible.